عن تصميم لغة البرمجة العربية "إبداع"

م. وائل حسن محمد علي

بسم الله و الصلاة و السلام علي رسول الله و علي آله و صحبه و مَن والاه، أما بعد:

فهذا الكُتيِّب هو ما طُلِب مِني كتابته في خلال منحةٍ بحثية (التي فزتُ بها خلال مشاركتِي في الموسم الرابع من برنامج المسابقات العلمي التليفزيوني "نجوم العلوم stars of science"). و كان ذلك بإشراف " د. كريم درويش" في معهد قطر لأبحاث الحوسبة QCRI = qatar computing research institute، في خلال الفترة من الأول من أبريل لعام 2013 م حتي الخامس من مايو لنفس العام.

و كانت المهمات المطلوبة مني تتلخص في التالي:

- إعداد تقريرٍ شاملٍ يتضمن الأمور التقنية التالية:
  - توضيح الغرض الأساسي من لغة البرمجة العربية "إبداع" و أهداف بنائها.
  - توضيح الأعمال البرمجية التي يُفترَض التمكن من إتمامها باستخدام لغة "إبداع".
  - توضيح القرارات التصميمية التالية:
    - هل لغة "إبداع" ستكون مُفسِّرة interpreted أم مُترجَمة compiled ؟.
    - هل ستكون "إبداع" لغةً ذات نظام أنواعٍ ثابت static typing أم نظام أنواعٍ مُتغيِّر dynamic typing ؟.
    - هل ستكون اللغة كائنيةً object oriented أم إجرائية procedural أم وظائفية functional ... إلخ ؟.
    - ما هي إمكانية استخدام الأكواد المكتوبة بلغاتٍ أخري في مشاريع مكتوبةٍ بإبداع ؟.
    - هل البرامج التي يتم كتابتها بإبداع يمكن أن تكون ذات خيوط تنفيذٍ متنوعة multi-threaded أم لا يمكن أن تكون إلا ذات خيط تنفيذٍ واحدٍ single threaded؟.
    - ما هي طريقة إدارة الذاكرة memory management: تعاملٌ آلي implicit garbage collection أم تعاملٌ يدوي explicit garbage collection ؟.
    - كتابة القواعد الكاملة للغة "إبداع".
- بناء واجهةٍ تفاعليةٍ interactive interface للمُفسِّر القياسي "أُبْدِع"، و وضعها علي شكل صفحة إنترنت يمكن تشغيلها من أي متصفح web browser بدون الحاجة لتنصيبها install.

و قد انتهيتُ بفضل الله تعالي من الجانب النظري، بينما لم ينته الجانب العملي بأكمله، و كذا بقي إجراء اختباراتٍ مكثفةٍ عليه لضمان كفاءته. و هذا الكُتيِّب ليس إلا الجزء النظري بعد حذف القسم الخاص بشرح قواعد لغة "إبداع" و كثيرٍ من شروحات القرارات التصميمية المختلفة؛ و ذلك لأن محل تلك القواعد و الشروحات في كتاب "رسالة البرمجة بإبداع"، و ليس من الجيد تكرار جزءٍ ضخمٍ كهذا في أكثر من مُؤلَّف.
و ربما أقوم فيما بعد بإذن الله تعالي بإضافة الشروحات التي تُوجد هنا فقط إلي كتاب الرسالة؛ حتي يظل علي

حالته التي أردتُها له في أن يكون مرجعاُ شاملاً لكل ما يخص لغة إبداع.

هناك وجهات نظرٍ مختلفة للغات البرمجة من حيث التعريف و الهدف، و من وجهة نظري الشخصية فإن لغات البرمجة هي أي وسيلةٍ يمكن أن يُخاطِب بها البشرُ الآلاتَ القابلة للبرمجة؛ لجعلها تقوم بالأمور التي يرغب البشر فيها. و بالاعتماد علي هذا التعريف فإن أشكال لغات البرمجة كثيرةٌ جداً؛ فهكذا لن تكون مقتصرةً فقط علي لغات البرمجة النصية (و هو الشكل التقليدي للغات البرمجة، مثل java و python) و الرسومية (و هو نوعٌ اقل انتشاراً، مثل DRAKON و CiMPLE)؛ بل ستضم داخلها كل أشكال "إعطاء الأوامر" و"التوجيه" من المُستخدِم للآلات القابلة للبرمجة.

و ما قد يكون غريباً جداً (و ربما مُستقبَحاً) من وجهة نظر الأغلبية أن هذا التعريف سيَضم إلي قائمة لغات البرمجة أشياء مثل siri و google voice search و ما شابههن من تقنيات التفاعل الحاسوبية مع الأصوات البشرية !، بل و سيَضم كذلك الحركات و الإشارات التي يمكن للحواسيب ترجمتها إلي أفعالٍ معينة (أعني Pointing device gesture)، و إن أصبح بالإمكان برمجة الحواسيب لفهم لغة الإشارة للصم و الكم فإن هذا التعريف للغات البرمجة سيضمها تلقائياً !

و من خلال ذلك التعريف فإن الهدف الأساسي للغات البرمجة هو أن تكون وسيط تفاهمٍ و تناغمٍ بين البشر و الآلات القابلة للبرمجة، و بما أن أهداف استخدام البشر لتلك الآلات يختلف اختلافاً كبيراً جداً من شخصٍ لآخرٍ و من تخصصٍ لآخر: فإن أشكال و صفات و مناطق القوي و الضعف تتغير من لغة برمجةٍ لأخري؛ تبعاً لتغير الحاجة التي تم تصميم و بناء كل لغةٍ من اللغات لها. كما أن البشر يختلفون فيما بينهم في كيفية القيام بالأمور المختلفة، و يحاول كل منهم أن يُبدِع طريقةً أفضل (بالنسبة له أو بالنسبة للعامة) لفعل الأمور التي يرغب في استخدام الآلة القابلة للبرمجة للقيام بها، و هذا سببٌ آخرٌ للاختلاف الواضح بين لغات البرمجة المتنوعة.

كما أن الأجهزة القابلة للبرمجة ذاتها تختلف في التكوين و الخصائص و طرق التعامل معها، و هذا يتسبب في اختلاف لغات البرمجة القابلة للاستخدام مع كل نوعٍ منها بكفاءةٍ و قوة، و بما يعكس الغرض الحقيقي من وراء تلك الآلات. فآلاتٌ مثل الروبوتات robots تركز علي التزامن و التوافق بين حركات الأطراف المختلفة و بين القراءات التي يتم الحصول عليها من المستشعرات sensors الخاصة بالجسد الآلي.

و لذلك نري أن أمثال هذه الآلات يكون لها لغات برمجةٍ تختلف نوعاً ما عن اللغات المعتادة لبرمجة الحواسيب الشخصية، مثل Action description language و Urbiscript. و علي الرغم من أنه بإمكاننا استخدم لغاتٍ عاديةٍ (أعني لغات البرمجة عالية المُستوَي عامّة الأغراض) لبرمجة

الروبوتات: إلا أننا علي الأقل سنحتاج إلي مكتباتٍ و أدواتٍ خاصةٍ بهذا الغرض؛ للتعامل مع تنفيذ العمليات المختلفة علي التوازي، مثل Microsoft Robotics Developer Studio و الذي يحتوي علي العديد من الأدوات التي من أهمها ما يُعرَف بالـ Concurrency and Coordination Runtime أو الـ CCR اختصاراً، و Decentralized Software Services (تُختصَر لـ DSS).

و هناك مثلاً المُمُتحكمات الميكروئية micro-controllers و الأنظمة المُضمَّنة embedded systems و التي تنتمي برمجتها إلي البرمجة منخفضة المستوي، و فيها نستخدم لغات التجميع assembly languages و التي (بطبيعة الحال) تُعتبَر هي أدني مستويات البرمجة التي يمكن الوصول إليها (لأنها ليست إلا لغة الآلة مكتوبةً بحروفٍ بشرية). لكن في نفس الوقت يمكن استخدام لغات برمجةٍ أعلي قليلاً في المستوي مثل الـ PL/M و الـ jalv2 و الـ C للقيام بذات الدور (قد يتطلب الأمر وجود بعض أكواد التجميع في البرامج)، و من الممكن كذلك استخدام لغاتٍ عالية المُستوَي عامة الأغراض لفعل ذات المهمة، مثل ada و java و lua أيضاً (رغم أن هذا قد يتطلب وجود نوعٍ من الطبقة الإضافية).

و هناك أيضاً بعض اللغات التي تُعتبَر مجرد تغييراتٍ للغاتٍ عالية المُستوَي بما يتوافق مع البرمجة منخفضة المُستوَي، مثل لغة الـEC++ التي تُعتبَر اقتطاعاً subset من الـC++ (اسمها اختصارٌ لـ Embedded C++). و فيها تم حذف العديد من الصفات الزائدة عن الحد المطلوب في علو المُستوَي مثل: الوراثة المُتعددة multiple inheritance و القوالب templates.

و مثل ذلك أيضاً لغة الـ pascal المُستخدَمة في مُترجِم Turbo51 و التي أضافت ميزاتٍ أخري علي الـ pascal الأصلية لتستطيع التعامل مع البرمجة منخفضة المستوي.

***

المهام التقليدية التي تقوم بها لغات البرمجة تختلف من لغةٍ لأخري بسبب اختلاف الهدف الأساسي من وراء بنائها، فمثلاً لغةً مثل الـ BASIC-256 هي لغةٌ تعليميةٌ في المقام الأوحد، و لذلك فإن أهم أهدافها ستكون:

- تصغير عدد قواعد اللغة إلي أدني حدٍ ممكن، و كذلك تسهيل هذه القواعد لأقصي درجة؛ فالهدف الرئيس هنا هو تعليم المفاهيم الأساسية في عالم البرمجة و ليس الاحتراف. و لذلك مثلاً نجدها لا تحتوي علي مكوناتٍ متقدمةٍ مثل الأصناف classes و لا الأغلفة properties و ما شابههن.
- الاعتماد علي الكلمات قدر الإمكان و البعد عن العلامات الغريبة قدر الاستطاعة؛ لأن الأطفال و المبتدئين لا يستخدمون العلامات بنفس الإفراط الذي هي عليه في البرمجة، أما الكلمات فيتقن الجميع التعامل معها، و عند اختيار الكلمات المُعتادة (أو غير المُستثقَلة) فإنه يصير أسهل بكثيرٍ جداً تقبُّل المفاهيم البرمجية، علي عكس الحالة التي يتم فيها الاعتماد علي العلامات الغريبة علي المبتدئ؛ فهنا تظهر مشكلةٌ إضافية تُضاف إلي العبء الذي يحمله المُمُعلِّم: حيث يتوجب عليه إفهام المبتدئ

معاني العلامات و الوظائف التي تقوم بها في البرامج، ثم يجب عليها جعله يعتاد وجودها أمامه و التعامل معها بالقراءة و الكتابة.

أما لغةٌ مثل الـ object pascal فبسبب أنها مُصمَّمةً في الأصل لعالم المحترفين فإنها لا تأبه كثيراً بمسألة التسهيل و التخفيف في قواعدها، و أهم أهدافها ستكون:

- الاحتواء علي المكونات التي يحتاجها المبرمجون المحترفون لبناء تطبيقاتهم. و نظراً لأنها لغةٌ عامّة الأغراض general purpose تُستخدَم في كثيرٍ من التخصصات: فقد أدّي هذا أن تكون كثيرة المُكوِّنات (ربما إلي درجة التُّخمة!). و لأنها تهتم بأن تكون قابلة للاستخدام في المجالات المختلفة مثل: البرمجة علي المستوي الأدني low level programming، و برمجة الطبيقات الاحترافية ذات الواجهات الرسومية السهلة الجميلة. لذلك فهي فهي تمزج بين المكونات التي تجعل بالإمكان استخدامها في كل تلك الغراض، مثل الـ union الذي يسهل لها الاستخدام في البرمجة منخفضة المستوي، و الأصناف و الواجهات interfaces التي تسهل استخدامها في بناء التطبيقات عالية المستوي و خاصةً ذوات الواجهات الرسومية.
- وجود مكتباتٍ قويةٍ تسهل بناء التطبيقات الضخمة بأقل درجةٍ ممكنةٍ من الجهد و العمل من جانب المبرمج المستخدم للغة. و قد أدّي هذا إلي أن تكون اللغة قابلةً للتمديد extensible علي العديد من المستويات، و من هذا أنه من الممكن إضافة المزيد و المزيد من جانب المستخدم لبناء مكتابته الخاصة.

و هناك لغاتٌ متخصصةٌ في مناطق معينة مثل الـ shell script الذي يختص بالتحكم في وظائف أنظمة التشغيل شبيهة اليُنِكْس unix-like ببساطة و سرعة. و علي الرغم من أن البعض (أظنهم قليلون جداً) يعتبره لغة برمجةٍ عامة الأغراض إلا أنه في واقع الأمر ليس كذلك علي الإطلاق. فالهدف الرئيس من ورائه هو ما قلناه سابقاً من التحكم في أنظمة التشغيل شبيهة اليُنِكْس ليس إلا. و لذلك فله الصفات التالية:

- غير محمول not portable، بحيث أنه يكاد يكون من المستحيل أن تقوم بكتابة بريمج script يقوم بأمورٍ معينةٍ علي اليُنِكْس ثم تقوم بتشغيله بشكلٍ عاديٍ علي الـ windows.
- صياغة قواعده ليست بالسهولة التي عليها بقية اللغات عالية المستوي مثلاً؛ و ذلك لأنه يُسرف في استخدام العلامات و الكلمات ذات المعاني الخاصة؛ و ذلك بغرض جعل عملية التكويد أسرع و أبسط (أود أن أسجل اعتراضي الشديد علي هذا الظن).
- لا يحتوي علي كثيرٍ من المكونات التي لا يحتاج إليها للقيام بمهمته الوحيدة. فمثلاً لا يحتوي الـ shell script علي مُكوّنٍ مثل المؤشرات pointers التي تستخدم في اللغات التي توجد بها للبرمجة علي المستوي المنخفض.

و علي الرغم من الاختلاف الواضح بين كل تلك الموا صفات المختلفة للغات البرمجة: فإنه يمكننا الجمع بين كل هذه الصفات الخا صة في مجموعةٍ من المعايير العامة، و هي كما يلي:

- الاحتواء علي الأ دوات التي تكفي للقيام بالأهداف الخا صة بتلك اللغة.
- عدم الاحتواء علي المكونات التي لا تخدم الهدف من وراء بناء اللغة.
- التوازن بين سهولة القواعد و قوتها بحيث يتم الميل إلي الجانب الأهم لأهداف اللغة من وجهة نظر مصممها. فلو كانت احترافيةً يتم الميل أكثر إلي القوة و لو علي حساب السهولة، و لو كانت تعليميةً فيتم الميل إلي جانب السهولة مع عدم الاهتمام بالقوة المفقودة.

و بما أن إبداع هي لغةٌ عامة الأغراض عالية المستوي فعند تطبيق القواعد السابقة عليها سنجد أنها:

- حاولت احتواء كل المكونات التي يحتاجها المبرمج للعمل في مستويً منخفض و مستويً عالٍ، و في نفس الوقت حاولت أن تكون بسيطةً و خفيفة القواعد قدر الإمكان؛ لأنها تحاول أن تكون تعليميةً في ذات الوقت. مما أدي إلي أن عدد قواعدها و مُكوّناتها قليلٌ جداً بالمقارنة مع لغاتٍ مثل الـ++C و الـ visual basic و الـ#C.
- لم تحتو إبداع علي الكثير من المكونات التي رأيتُ أنها لا لازمة لها بالنسبة للغة برمجةٍ عالية المستوي عامة الأغراض، مثل المؤشرات pointers و الواجهات interfaces و غيرهن مع الاستغناء عنهن بضم مكوناتٍ مختلفةٍ يمكنها القيام بذات الأمور و لكن علي نحوٍ أبسط و/أو أسرع و/أو أقوي.
- أغلب قواعد إبداع سهلةٌ جداً و منطقية و نظيفة، إلا أن هناك الكثير من القواعد التي كان من اللازم أن تكون معقدةً لأنه لم يكن هناك وسيلةٌ للحصول علي قوتها من غير الاختيار ما بين أمرين:
  - العبث بالمكونات الأخري و تخريبها،
  - أو تعقيد تلك القواعد فقط و ينتهي الأمر.

و بالطبع فقد أخذتُ القرار الثاني و رأيتُ أنه لا مشكلة في أن تكون بعض القواعد المتقدمة معقدةً بعض الشيء؛ ففي كل الأحوال ليس من المفترض أن يدرس المبتدئون هذه القواعد و المحترفون الذين سيستخدمونها لن يجدوا صعوبةً في فهمها و تطويعها للاستفادة منها إلي المُمُنتَهَي.

شرح القرارات التصميمية للغة إبداع

ا ستخدام أكوا د اللغات الأ خري في إبداع

هناك أكثر من حالةٍ لهذا الأمر:

- ملفات مكتبات الربط الديناميكي (مثل الـ DLL في الويندوز و الـ SO في اِلِقْنُو/لِينُكْس GNU\linux) و التي هي عبارة عن أكوا دٍ مكتوبةٍ بإبداع ثم تم ترجمتها لكي تكون مكتبة مُتنقلةً قابلةً للا ستخدام في زمن التشغيل:

  و هذه تُعامَل كما تُعامَل ملفات الباقات العا دية تماماً، أي نستخدم معها الأمر أضم، ثم تُستَخدَم مُكوِّناتها الدا خلية كالعا دة.

  (لم يتم بناؤها في الإ صدارة الأ خيرة 1.10 من مُفسِّر اللغة).
- واجهات أنظمة التشغيل المُختلفة API التي ربما يو د المُبرمِج الو صول إليها في برامجه: و هذه لم يتم الإ ستقرار علي الموقف منها في إبداع (في هذه الإ صدارة من المُفسِّر علي الأقل).
- تصميم و بناء طريقةٍ تسمح با ستعمال أكوا د مكتبة الـjdk من دا خل لغة إبداع بدون الحاجة لقواعدٍ مُعقدةٍ لهذا، فيصير الأمر سهلاً كأنك تستخدم أكوا داً مكتوبةً بلغة إبداع ذاتها ! و يصبح من الممكن أن تكتب كو داً كالتالي علي سبيل المثال:

```
صنف1 كائن1 = صنف1()


صنف صنف1 يرث jdk_javax_swing_Box صنف2:
    \ كود الصنف صنف1 /
```

و هذا يعني قفزةً هائلةً في قدرات إبداع ؛ لأنها ستكون قا درةً علي برمجة المشاريع الضخمة بفضل قدرتها علي الو صول لمكتبة الـjdk بكل ما فيها من إمكانياتٍ جعلتها واحدةً من أكبر المكتبات البرمجية إطلاقاً !، و أيضاً ستظل إبداع في أقصي درجات المحمولية لأن مكتبة الـjdk محمولةٌ للغاية.

كما يعني أنه سيكون بالإمكان بناء المكتبة القيا سية لإلإبداع بالاعتما د علي مكتبة الـjdk مؤقتاً؛ فرغم قدرات مكتبة الـjdk الكبيرة إلا أنها نظراً لأنها مكتوبةٌ بلغة الـjava فهي تحوي النقائص التي فرضتها عليها، مثل عدم قدرة الـfunctions علي إعا دة سوي خرجٍ واحدٍ فقط، بينما في إبداع يمكن للإجراء إرجاع أي عددٍ يرغب فيه من المخرجات، كما أنه ليست هناك خا صيتا تحديد القيم

الاقتراضية للمُدخَلات و المُخرجَات في الإجراءات. كما أنه ليس هناك ما يشبه الإجراءات الحرة في الـjava و لا يوجد مبدأ الوراثة المتعددة، و غيرهن من الأمور التي تجعل من الأفضل بناء طبقةٍ برمجيةٍ صغيرةٍ نبني من خلالها المكتبة القياسية كغلافٍ wrapper للـjdk. و فيما بعد بإذن الله عز و جل نقوم ببنائها بالكامل من الصفر.

و هذا الأسلوب في العمل له فائدةٌ أخري هي زيادة سرعة بناء المكتبة القياسية و سهولة التعديل فيها بكل بساطة، و هو ما يعني أن تصميم المكتبة يمكن تحسينه بشكلٍ أبسط و أسرع بكثيرٍ مما كان الحال لو كنا قد بنيناها من الصفر.

single threaded vs. multi-threaded

إبداع لغةٌ تدعم الخيوط المُتعددة لا الخيط الواحد؛ بسبب أن ذلك يجعل الأداء يتضاعف عن الحالة التي يتم فيها الاعتماد علي خيط تنفيذٍ واحد. صحيحٌ أن هذا سيجعل قواعد اللغة أكبر بقليل و أكثر تعقيداً عما كانت لتكون عليه بدون تعديد خيوط التنفيذ: إلا أن المردود في كفاءة البرامج و حسن استغلالها لموارد العتاد المُتاح لها يشفع للعيوب السابق ذِكرها.
كما أن هناك أنواعٌ من التطبيقات لا يمكن بناؤها إلا باستخدام الخيوط المتعدد. و بما أن إبداعٌ لغةٌ عامّة الأغراض فليس من الجيد لها أن تُهمِل هذا النوع من التطبيقات لمجرد أنه سيجعل عدد قواعدها أكبر قليلاً.

strongly typed or not

إبداع لديها قواعدٌ صارمةٌ للتحويل بين أنواع المتغيرات المختلفة و كيفية مزجها في تعبيرٍ expression واحد. فعلي سبيل المثال لا يمكنك التحويل بين النوع النصي و النوع الرقمي إلا باستخدام إجراءٍ قياسيٍ خاص، مثل:

```
أكتب.نص.سطر("قيمة الرقم = " + إلي.نص(10))
```

و عند الرغبة في تحويل قيمة كائن لقب enumeration إلي نصٍ نقوم بفعل ما يُشبِه التالي:

```
يوم يوم.الإجازة = يوم_جمعة
أكتب.نص.سطر("يوم الإجازة هو " + يوم_إلي.نص(يوم.الإجازة))
```

```
لقب يوم :
    سبت أحد اثنين ثلاثاء أربعاء خميس جمعة
```

و هكذا.

و لكن من الممكن أن يقوم المبرمج بتحييد هذه الخواص إذا احتاج لذلك عن طريق:

- استخدام التنويع المُتغيِّر dynamic typing المُتاح في إبداع، و يكون هذا عن طريق استخدام النوع حر كما سيلي شرحه في المواصفات القياسية لإبداع بإذن الله تعالي. و هذا مفيدٌ في التطبيقات التي لا يهتم فيها المبرمج بنظام الأنواع و لا تكون هناك أي حاجةٍ للتحسينات التي تتعلق بالذاكرة أو المعالج. و كذلك في بعض الحالات التي لا يمكن تجاوزها إلا بنوعٍ من التنويع المتغير أو ما يُحاكيه (مثل استخدام الصنف Object في لغة الـ java).
- استخدام لغات التجميع في الإجراءات الخاصة للوصول إلي البايتات التي يتكون منها المتغير؛ و من ثم يكون بإمكانه أن يقوم بفعل ما يرغب فيه بغض النظر عن أي اعتبارٍ "عالي المستوي". و هذا مُفيدٌ في الأمور التي يتم برمجتها في المستوي المنخفض.